UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Evaluación de prestaciones de Aplicaciones en Internet

Pablo Castedo Sanjuán
Tutor: Javier Aracil

JUNIO 2021

Evaluación de prestaciones de Aplicaciones en Internet

AUTOR: Pablo Castedo Sanjuán
TUTOR: Javier Aracil

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2021

Resumen

Este Trabajo Fin de Grado nace de la necesidad actual de desarrollar tecnologías estables y seguras, si bien es cierto que este precepto es aplicable en cualquier sector relacionado con el desarrollo de productos, tanto hardware como software, este proyecto está centrado en resaltar cómo es de importante que los servidores que proveen servicios online puedan ser mantenidos y estabilizados con el fin de proteger a los clientes que hacen uso de ellos.

Gran parte del contenido que nos encontramos en internet, por no decir la mayoría, se encuentra alojado en servidores y en una época anterior donde dependíamos de este producto de una forma más ocasional, las caídas de servidores era algo permisible y de consecuencias relativamente aceptables, sobre todo en comparación con lo que estos hechos significan actualmente, donde un fallo de seguridad o la más mínima situación inesperada, puede llevar a que millones y millones de usuarios pierdan la capacidad de realizar tareas que se han convertido en parte de su vida diaria.

Además, estas tareas no solo engloban las actividades de ocio que los usuarios puedan llevar acabo, sino que en la actualidad internet se ha convertido en un pilar básico de nuestra sociedad, donde las empresas, las instituciones, los estados y demás órganos se apoyan para cumplimentar sus labores mediante el uso de una infinidad de variados servicios. Ejemplo de esto son los hechos ocurridos el 8 de junio de este mismo año (2021) donde la caída del CDN Fastly se tradujo en la inutilización de una gran parte de internet tal y como lo conocemos, dejando inoperativos servicios como GitHub, Imgur, HBO, Twitter, Reddit, Twitch y muchos otros.

Sentadas estas bases, ya se puede proceder a leer el documento entendiendo la intención inicial del autor. En el proyecto se explorarán los fundamentos teóricos que lo componen y se discutirán distintas alternativas tanto de forma teórica como mediante la realización de un ejemplo práctico y la exposición de los resultados obtenidos.

Palabras clave

- WWW (World Wide Web)
- Servidores
- Grafana
- HTTP (Hypertext Transfer Protocol)
- Clientes
- Red
- LAN (Red de Área Local)
- Protocolos
- Tráfico de red
- Firewall
- Java
- Python

Abstract

This Bachelor Thesis is born from the current need to develop stable and safe technologies, although it is true that this precept is applicable in any sector related to the development of products, both hardware and software, this project is focused on highlighting how important, is that the servers that provide online services can be maintained and stabilized in order to protect the clients who make use of them.

Much of the content that we find on the internet, if not most, is hosted on servers and in a previous era where we depended on this product in a more occasional way, server crashes were something permissible and of relatively acceptable consequences, especially in comparison with what these events mean today, where a security breach or the slightest unexpected situation, can lead to millions and millions of users losing the ability to perform tasks that have become part of their daily lives.

In addition, these tasks not only include leisure activities that users can carry out, but today the Internet has become a basic pillar of our society, where companies, institutions, states and other bodies support each other. to complete their tasks using an infinity of varied services. An example of this are the events that occurred on June 8 of this year (2021) where the fall of the Fastly CDN resulted in the disablement of a large part of the internet as we know it, leaving services such as GitHub, Imgur, HBO inoperative. , Twitter, Reddit, Twitch, and many others.

Once these bases have been established, you can now proceed to read the document understanding the initial intention of the author, in the project the theoretical foundations that compose it will be explored and different alternatives will be discussed both theoretically and through the realization of a practical example and the exhibition of the results obtained.

Keywords

- WWW (World Wide Web)
- Servers
- Grafana
- HTTP (Hypertext Transfer Protocol)
- Clients
- Net
- LAN (Local Area Network)
- Protocols
- Net traffic
- Firewall
- Java
- Python

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

1.1. Motivación

Hoy en día gran parte de las aplicaciones dispuestas en internet, por no decir la mayoría, se disponen siguiendo una arquitectura **cliente-servidor**, sobre todo para los usuarios no especializados de este servicio, los cuales representan casi el total de la población activa que a diario realizan diversas actividades a través de la www.

Global Rank	Domain	Monthly visits (billions)	Parent	Country
1	Google.com	60.49	Alphabet Inc	United States
2	Youtube.com	24.31	Alphabet Inc	United States
3	Facebook.com	19.98	Facebook, Inc	United States
4	Baidu.com	9.77	Baidu, Inc	China
5	Wikipedia.org	4.69	Wikimedia Foundation	United States
6	Twitter.com	3.92	Twitter, Inc	United States
7	Yahoo.com	3.74	Verizon Comm. Inc	United States
8	pornhub.com	3.36	Mindgeek	Canada
9	Instagram.com	3.21	Facebook, Inc	United States
10	xvideos.com	3.19	WGCZ Holding	Czech Republic

Tabla 1: Sitios web más visitados

Ejemplo de ello es que como se puede observar en esta figura el top 10 de sitios más visitados en internet se compone exclusivamente de páginas enfocadas en un modelo Cliente/Servidor.

Teniendo en cuenta los datos expuestos resulta imposible no pensar en cómo de importante es que esta arquitectura, siendo el canal por el que pasan cientos de billones de peticiones a diario, sea un medio no solo eficiente, sino además seguro, tanto como para los clientes que realizan las peticiones como para los poseedores de los servidores que tramitan lo que los clientes requieren.

Si bien es cierto que la robustez de internet ha ido mejorando, también es cierto que la cantidad de datos y clientes que lo utilizan ha aumentado a igual ritmo, lo que conlleva a que las caídas de servicios sea algo más que común. Servicios web utilizados a diario como Whatsapp (febrero de 2014, deja sin servicio a 450 millones de usuarios), Sony PSN (sufrió ataques DDos que dejaron la red inutilizable por más de seis días), aun siendo servicios pertenecientes a empresas grandes, con amplia cantidad de recursos, se ven afectados por estos problemas y las pérdidas ocasionadas pueden ser realmente devastadoras para sus respectivas economías y reputaciones.

Por lo tanto la motivación principal y la intencionalidad detrás de este trabajo realizado, no es otra que la de demostrar la importancia de que estos servidores, que hacen las veces de cimientos para la red más usada a nivel mundial, sean máquinas seguras e incluso en el caso de que por cualquier razón (actos maliciosos, tráfico descontrolado/imprevisto, etc) estos servicios fueran a caerse o se cayeran, sus propietarios tuvieran los medios y métodos para prever esta situación y/o estar al tanto de ella lo antes posible para solucionarla.

Por último y por si los datos expuestos no son suficiente prueba de la importancia del manejo de estas situaciones, ya existen herramientas que recopilan y almacenan datos y sucesos de esta índole, por ejemplo, la página Downdetector [16], donde podemos encontrar un historial detallado y mensual de caídas registradas, así como información de las causas de estas.

Outages overview

June 2021

Company	Date	Started
Zoom	06/09/2021	10:38 a.m.
Sling	06/09/2021	10:17 a.m.
Windstream	06/09/2021	10:06 a.m.
Coinbase	06/09/2021	9:24 a.m.
Earthlink	06/09/2021	9:16 a.m.
Amino Apps	06/09/2021	9:08 a.m.
Quizlet	06/09/2021	9:05 a.m.
WOW	06/09/2021	7:50 a.m.
League of Legends	06/09/2021	6:09 a.m.
Capital One	06/09/2021	6:04 a.m.
Instagram	06/09/2021	4:49 a.m.
PG&E	06/09/2021	4:21 a.m.
Spectrum	06/09/2021	3:33 a.m.
Roblox	06/09/2021	3:13 a.m.
Cox	06/09/2021	3:12 a.m.
Runescape	06/09/2021	3:05 a.m.
TDS Telecom	06/09/2021	2:09 a.m.
Xbox Live	06/09/2021	1:46 a.m.
Call of Duty	06/09/2021	1:44 a.m.
Blizzard Battle.net	06/09/2021	1:44 a.m.
AT&T	06/09/2021	1:32 a.m.

Tabla 2: Ejemplo de un registro de caídas de servicios [16]

1.2. Objetivos

El objetivo por tanto de este documento es proveer de un método probado y demostrado para la monitorización de servidores, mediante el cual sea posible hacer un correcto seguimiento de factores, que tienen relación con la estabilidad de un servidor y que nos pueden ayudar a identificar cuando pueden estar ocurriendo situaciones que pueden representar un peligro para la estabilidad de nuestro servicio.

Además, se ha tenido muy en cuenta que este método sea sencillo y sobre todo reproducible con la mayor facilidad, por lo tanto, será tanto explicado el proceso como proporcionados los códigos y herramientas que se han desarrollado para las convenientes pruebas realizadas. Herramientas de las cuales se hablará en detalle más adelante, que han sido estudiadas y elegidas debido a múltiples criterios, ¿Es una herramienta usada popularmente por usuarios/empresas?, ¿Es una herramienta probada de ser eficiente?

Por lo tanto y para sintetizar esta información, los objetivos que se han tratado de lograr son:

- Lograr una correcta comprensión de la arquitectura Cliente/Servidor, sus ventajas, así como sus limitaciones.
- Recrear un servicio online y probar su robustez a la vez que recopilar datos de interés de este.
- Dejar sentadas unas firmes bases, para motivar futuros proyectos relacionados con este, que aprovechen la sinterización de la información recopilada para su correcta recreación.

1.3. Organización de la memoria

La memoria consta de los siguientes capítulos:

- Primera parte: Diseño y debate previo sobre el Software a utilizar

En este capítulo se exponen las ideas y valoraciones realizadas previamente de desarrollar el entorno de pruebas, el motivo por el cual se ha decidido utilizar el Software designado y que alternativas se han planteado durante la investigación inicial.

- Segunda parte: Implementación del Software elegido

Tras elegir y asegurarse de cuál va a ser el método a seguir, en este capítulo se enseña cómo el proyecto ha sido desarrollado e implementado, que problemas han surgido y cuales han sido las soluciones o qué variaciones respecto a la idea inicial han tenido que ser aplicadas.

- Tercera parte: Resultados obtenidos

En este capítulo se realiza la exposición, muestra y análisis de los datos y resultados que se han generado con el desarrollo y ejecución del Software creado, además de valoraciones objetivas respecto al producto creado.

- Cuarta parte: Conclusiones

Por último, se realiza una valoración respecto a los resultados obtenidos, importancia de los datos analizados, utilidad y posible proyección y mejora de los mismos.

2. Estado del arte

2.1. Arquitecturas más comunes en internet

Por arquitecturas, entendemos las formas o maneras en las que se organizan y esquematizan los usuarios de una red, normalmente estas arquitecturas son implementadas atendiendo a ciertos criterios, como puede ser la cantidad de clientes esperada o la manera en la que se espera distribuir y gestionar tanto la información como los datos a través de la red.

Si bien es cierto que la arquitectura que va a ser explorada y probada en profundidad es, como se ha comentado en la motivación, la arquitectura cliente-servidor, es importante hacer una vista general para definir y entender porque este modelo es el más utilizado en términos generales, así como qué alternativas existen, para que son usadas y qué ventajas e inconvenientes pueden tener comparando unas con otras.

Es posible que para casos concretos existan otras múltiples alternativas a las presentadas, sin embargo, es importante recalcar que las tres arquitecturas presentadas son las más conocidas y utilizadas para gestionar el tráfico en internet.

Para un buen entendimiento de las arquitecturas se facilitarán distintas figuras y serán posteriormente explicadas en detalle.

2.1.1. Arquitectura Cliente/Servidor

Esta será la arquitectura explorada con más profundidad debido a su relevancia actual, así como el resto de los motivos expuestos en la introducción del trabajo.

Este modelo organizativo se caracteriza por la existencia de dos agentes primarios, ambos interactúan entre ellos y comparten ciertos aspectos, pero cumplen funciones muy distintas.

El Cliente:

Los clientes son aquellos usuarios que realizan las peticiones, es decir, a través de múltiples protocolos y capas, tratan de realizar una conexión con un servidor de interés para este. Tras establecer la conexión, el cliente tiene la posibilidad de requerir al servidor que le proporcione un recurso, este recurso puede ser un archivo, el resultado de procesar datos enviados por el cliente, una compra, etc.

Es importante recalcar que generalmente, el cliente desconoce toda o casi toda la información acerca de otros clientes o de la misma estructura interna del servidor al que accede.

El Servidor:

El servidor, es el agente encargado de, en primer lugar, procesar las peticiones que reciba y en caso de ser posible, cumplimentarlas y devolver al cliente que haya realizado la petición, el recurso que haya sido requerido.

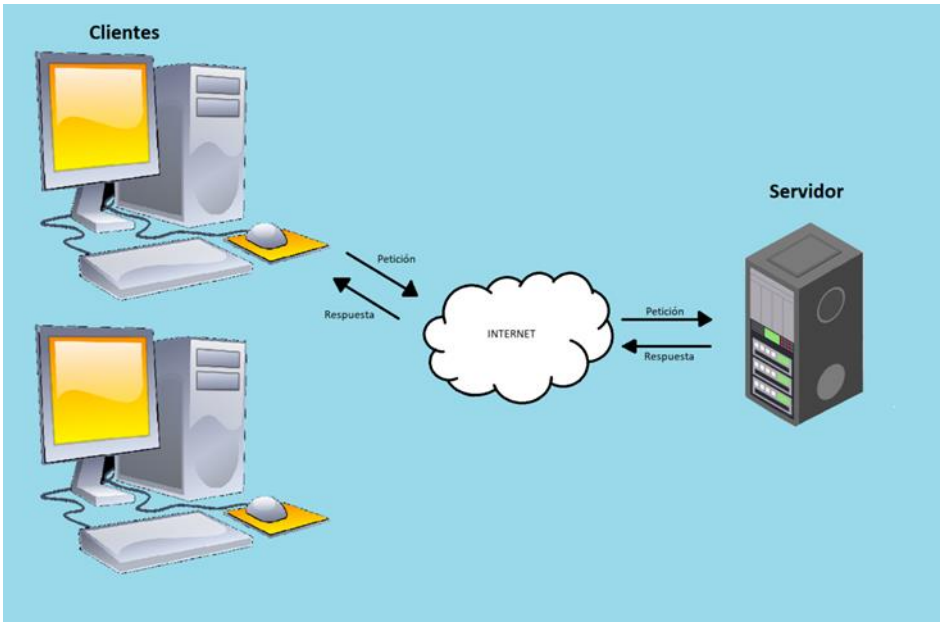

Figura 1: Ejemplo visual de arquitectura cliente/servidor

En la figura, se puede apreciar como el Cliente, realiza una petición, esta es procesada y enviada al servidor y este último, tras recibir la petición, genera una respuesta que es mandada de vuelta por la conexión establecida hacia el mismo cliente que realizó la solicitud.

Con esta información proporcionada, se pueden apreciar ciertos rasgos determinantes en esta arquitectura, el primero es que es un modelo cómodo para los clientes, que, con una información mínima, concretamente la dirección virtual del servidor, pueden cómodamente acceder a contenidos a través de la red, de forma rápida, eficiente y dependiendo del servidor, más o menos segura. Sin embargo, estas facilidades para los clientes, hacen que gran parte de las responsabilidades del proceso recaigan directamente sobre el servidor, que es encargado de, almacenar las funcionalidades que se ofrezcan, tanto si están dentro del propio servidor, como si el servidor se usa únicamente de intermediario para la comunicación con una base de datos, así como de gestionar, en la medida de lo posible y de la manera más eficiente una cantidad de peticiones, que pueden ir desde un rango que el servidor considere factible, hasta una cantidad exagerada de las mismas que en cierto momento y sobre todo si es de manera imprevista, puedan significar un problema en el estado del servidor.

El hecho de que todas estas responsabilidades recaigan sobre el servidor, simboliza que en caso de que se de cualquier problema, todo el servicio que el servidor ofrezca dejará

de estar operativo, tanto si este problema es algo físico (La máquina que aloja la funcionalidad del servidor deja de funcionar correctamente), problema que es solucionable por ejemplo mediante la aplicación de redundancia, haciendo que múltiples máquinas en paralelo alojen los datos y gestionen las peticiones, evitamos que la caída de una máquina derrumbe el servicio, aunque esta solución puede significar costes significativos tanto en el establecimiento de estas máquinas como en el mantenimiento de las mismas, como si el problema es causado por motivos más difíciles de controlar (Alto tráfico de peticiones, caída de un servicio intermedio ajeno al servidor, etc.)

Sin embargo, esta arquitectura plantea ciertas ventajas, sobre todo en comparación con la arquitectura peer-to-peer, que son motivo suficiente para que su uso sea tan extendido hoy en día [17]:

- La centralización de los datos tiene como resultado que el manejo de estos sea fácil y eficiente, así como el manejo de errores y problemas que puedan surgir.
- Generalmente, el hardware usado en los servidores está específicamente diseñado para la realización de esa función, lo que conlleva ventajas respecto a la eficacia y más que nada, a la eficiencia a la hora de procesar solicitudes de clientes.
- Los clientes se encargan de conocer la dirección del servidor, y el servidor solo debe devolver la respuesta a través de la misma conexión, esto aligera en gran medida el tráfico que pasa a través de la red, evitando posibles congestiones.

2.1.2. Arquitectura Peer-to-Peer

Este modelo, se plantea como alternativa directa y opuesta ante el modelo cliente-servidor, pues se basa en la descentralización de los datos y las responsabilidades que, en este caso, son distribuidas entre todos los clientes de la red en lugar de recaer en un solo agente.

Esta arquitectura plantea ventajas e inconvenientes respecto a su arquitectura contrapuesta y en lugar de componerse por dos agentes con funciones completamente distintas que se conectan, todos los miembros de la red denominados nodos, hacen las veces de clientes y de servidores.

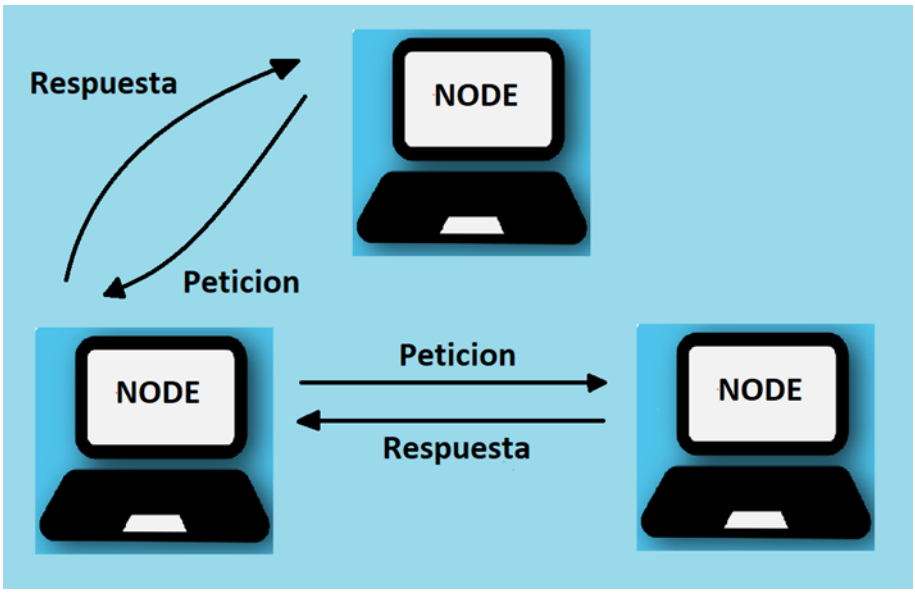

Figura 2: Ejemplo visual de arquitectura peer to peer

Como se puede observar en la figura, la estructura de la red no dispone de un ordenador central que organice a los usuarios y almacene información, ya sea datos de direcciones de otros clientes (en este caso nodos), o datos completos (archivos, por ejemplo), si no que los datos como tal están distribuidos en los nodos y no necesariamente de forma uniforme.

Por lo tanto los nodos de la red solo tienen una forma de encontrar la información que están buscando, teniendo en cuenta que todos los nodos tienen la capacidad de preguntar y responder a los nodos con los que tienen contacto (adyacentes), cada nodo tendrá como labor proveer de aquellos recursos que le sean requeridos, a la vez que dispondrá de la capacidad para iniciar una llamada progresiva que preguntará de nodo en nodo hasta encontrar el contenido deseado, el cual podrá ser requerido en caso de existir.

Esta metodología parece costosa, sobre todo a medida que aumenten los nodos en la red, sin embargo, la redundancia de datos y otras técnicas como tablas de fingers hacen a las redes p2p un modelo eficiente y usado en más servicios on-line de los que pueda parecer a priori. Tanto de forma pura como en redes p2p híbridas, podemos encontrarnos con esta arquitectura en páginas como Torrent, Netflix, Spotify o Skype.

2.1.3. Arquitecturas híbridas

Debido a las inconsistencias en cuanto al tráfico, así como a las dificultades que pueden encontrarse en las redes peer-to-peer a la hora de identificar y encontrar los nodos que sean útiles en cada momento para cada usuario, existe como alternativa la arquitectura híbrida.

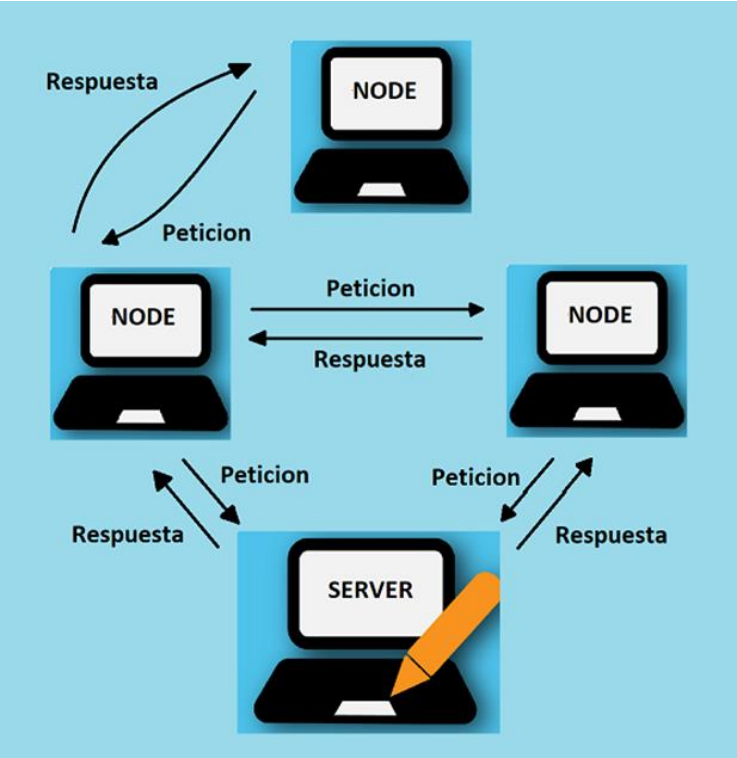

Figura 3: Ejemplo visual de una arquitectura híbrida

En estas arquitecturas los nodos están conectados entre ellos y siguen cumpliendo los roles de cliente y servidor, sin embargo, se ven beneficiados de poder conectarse a un nodo central, el cual tiene conexión con todos los nodos, cuya única función es la de dirigir el tráfico entre nodos para evitar congestiones e indexar nodos y contenidos para facilitar el acceso a estos mismos por parte de los clientes, lo que logra una mayor eficiencia.

Sin embargo, esta arquitectura, si bien es cierto que soluciona problemas de las dos organizaciones de red explicadas con anterioridad, también arrastra problemas de ambas, por ejemplo, de nuevo gran responsabilidad cae en el nodo central (Servidor) y si este sufre una caída la red volverá a sufrir desbalance de peticiones, y se convertirá en una red p2p pura con sus respectivos problemas:

- Los nodos no están diseñados específicamente para atender una gran cantidad de peticiones.
- Los nodos no están diseñados para mantener unos estándares mínimos de seguridad.
- Si los nodos comienzan a abandonar la red, esta se volverá inservible.

2.2. Protocolos de conexión

Los protocolos, comúnmente y no solo en este ámbito de las conexiones por internet, son reglas o convenios que se siguen de mutuo acuerdo para facilitar una correcta

comunicación, por ejemplo, y este es el principal motivo y definición de porque se usan como base en internet para enlazar usuarios, dispositivos, servidores y clientes, etc.

Sin un correcto uso de estos protocolos, la coordinación a través de la www sería imposible, debido a esta importancia y la base que constituyen tanto en internet como en este proyecto, un correcto entendimiento de estos es necesario y de nuevo como con las arquitecturas.

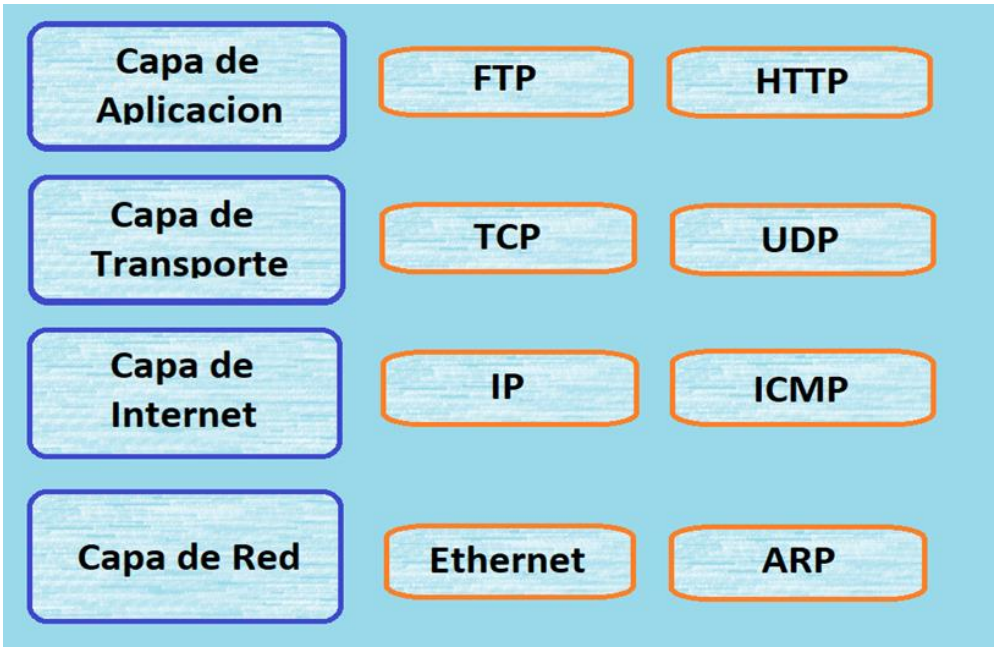

Figura 4: Disposición básica de las capas de Internet

Multitud de distintos protocolos son usados a través de las distintas Capas de internet:

- Capa de Aplicación: Es la capa más próxima al cliente final de aplicaciones en internet
- Capa de Transporte: Es la capa donde se gestionan las conexiones entre los hosts, a donde van los datos, a qué velocidad y otros aspectos de esta índole.
- Capa de Internet: Capa encargada de enrutar las conexiones y buscar el camino que siguen los datos desde una dirección hasta otra.
- Capa de Red: Capa que maneja el aspecto físico de las conexiones.

Existen distintas subcapas más específicas, pero es suficiente con tener una visión general de este esquema en mente para entender apropiadamente el funcionamiento de los protocolos en su contexto.

El protocolo que se explicará con mayor detalle es el usado durante el desarrollo del entorno de pruebas, el protocolo TCP/IP, pero distintos protocolos serán expuestos para entender porque se ha elegido este y no otro.

2.2.1. Capa de transporte

2.2.1.1. Protocolo TCP/IP

Este protocolo es el más usado para el intercambio de datos en internet y su nombre es divisible en los dos principales agentes explicativos del funcionamiento del protocolo.

IP o Internet Protocol, es la dirección que proporciona la localización de algo a través de internet. Esta dirección se compone por cuatro números separados por puntos entre el 0 y el 255, de la forma XX.XX.XX.XX. Esta representación plantea problemas, ya que la cantidad de combinaciones posibles asciende hasta 2554 de forma teórica, lo que se traduciría como 4.294.967.296 direcciones asignables pero, teniendo en cuenta que determinadas combinaciones tienen usos reservados como por ejemplo servir de direcciones de redes locales, este número que parece grande a priori se queda definitivamente corto a la hora de localizar usuarios y máquinas en internet, por lo que ha sido necesaria la creación de otros protocolos y métodos para expandir los usos de las direcciones.

TCP en cambio representa la metodología que se sigue a la hora de mandar la información, la cual se puede explicar como la división de la información que quiere ser compartida, en pequeños bloques denominados datagramas, estos bloques son mandados a través de la red con la información suficiente como para que un usuario receptor, haciendo uso del mismo protocolo TCP/IP, sea capaz de recomponer el mensaje, así como detectar errores en el envío en caso de que los hubiera.

El factor más representativo de este protocolo en contraposición al protocolo UDP, es que TCP es sin lugar a dudas más lento y pesado, sin embargo, el hecho de que asegure que la información va a ser recibida completamente y en el orden adecuado hace que su uso sea el más extendido, si bien es cierto que se ha de tener en cuenta su finalidad, por ejemplo aplicaciones que dependan de la integridad de la información (Gestores de correo, Gestores de archivos, etc.) usarán TCP/IP.

2.2.1.2. Protocolo UDP/IP

La mayor diferencia que presenta este protocolo respecto al protocolo TCP/IP es la conexión entre los agentes que se comunican en el proceso. Mientras que en TCP se controlan los datos que se mandan entre dos usuarios A y B, es decir, la conexión es establecida por ambas partes y el receptor puede pedir al emisor que vuelva a enviar los datos en caso de que la recepción de estos no haya sido efectiva, en UDP la conexión es unidireccional, el emisor A envía los datos al receptor B sin necesidad de una confirmación de que la conexión entre ambos es aceptada, la razón de esto es que en este protocolo, el receptor no conoce ninguna información del emisor a parte de la dirección IP, en contraposición con el protocolo TCP donde ambos agentes poseen información el uno del otro.

Que esta conexión entre ambos agentes no tenga que ser aceptada por ambos agiliza en gran manera el envío de datagramas. Debido a esto este protocolo es utilizado en aplicaciones que necesitan una gran velocidad de recepción de paquetes, como son aquellas cuya función es la reproducción de audio y video (servicios de streaming de contenido multimedia, servicios de videollamadas, etc.)

Sin embargo, la alta velocidad de este protocolo sacrifica a cambio otros aspectos, por ejemplo, la confianza que se tiene en que el contenido llegue correctamente, lo que lo hace un protocolo menos fiable para transmisión de datos específicos y además pueden darse múltiples errores al utilizarlo, como la pérdida de paquetes o distintas fallas de seguridad relacionadas con ataques DDos.

2.2.2. Capa de Aplicación

2.2.2.1. Protocolo HTTP

El protocolo HTTP (Hyper Text Transfer Protocol), es el protocolo de estructura cliente-servidor más usado hoy en día y también es el protocolo que se ha utilizado en el desarrollo de este trabajo.

HTTP es un protocolo que es de base fácil de interpretar incluso para usuarios sin un conocimiento técnico. Este se compone de dos mensajes, el primero mandado hacia el servidor por parte del cliente, se puede formular de múltiples maneras dependiendo de que quiera obtener como respuesta el cliente. De todos los parámetros que conforman las peticiones de los clientes, el más importante es el tipo de petición que es.

A continuación, se enumeran y describen con brevedad los dos tipos más básicos usados, aunque cabe remarcar que existen multitud de otros tipos:

Tipo **GET**:

GET /recurso.html

Este tipo de petición es usado para solicitar recursos al servidor, en este caso el recurso "recurso.html", también es posible hacer peticiones GET más complejas añadiendo argumentos adicionales, los cuales se añaden al final de la petición usando un símbolo de interrogación como separador de la URL y las variables, por ejemplo:

GET /recurso.html?variable1=valor1

Motivo que hace que esta petición sea muy utilizada, siendo esta legible y rápida de crear e interpretar, pero que a su vez hace que a su vez plantea el problema de mostrar la información que el cliente le manda al servidor, lo que la hace susceptible de ser interceptada, motivo suficiente para que este tipo de petición no sea utilizada en registros de usuario o cualquier otra situación que trate información delicada.

Tipo **POST**:

Este tipo de petición solventa los problemas presentados por la petición GET, en primer lugar, las peticiones tipo POST transmiten los dados anexionados al encabezado de la petición en un espacio conocido como cuerpo, que puede ser apreciado en la figura de ejemplo lo que evita que los datos sensibles puedan ser visibles en la misma petición y en segundo lugar, las peticiones tipo GET se ven limitadas por el máximo número de caracteres que puede tener una URL (2048) mientras que la cantidad de información que es potencialmente enviable en una petición tipo POST es ilimitada.

Además, en la figura expuesta se pueden apreciar otros ejemplos de qué información suele enviarse en los encabezados de las peticiones HTTP, como que versión del protocolo se usa, que tipo de contenido se requieren, cuál es el tamaño del cuerpo que se envía, etc.

```
POST /cgi-bin/create.pl  HTTP/1.1
Host: examples.ora.com
Referer: http://examples.ora.com/create.html
Accept: image/gif, image/x-xbitmap,
   image/jpeg, image/pjpeg, */*
Content-type: application/x-www-form-
   urlencoded
Content-length: 38

user=util-tester&pass1=1234&pass2=1234
```

Figura 5: Ejemplo de una petición POST.

Existen otros tipos de peticiones HTTP como por ejemplo el tipo HEAD, que únicamente pide al servidor la cabecera de la respuesta y por tanto puede ser usado para de manera ágil comprobar si cierta solicitud puede ser atendida, o el método OPTIONS que devuelve qué tipos de solicitud acepta el servidor para un determinado recurso.

Respuesta **HTTP** desde el servidor:

```
HTTP/1.0 200 OK
Date: Sat, 20-May-95 03:25:12 GMT
Server: NCSA/1.3
MIME-version: 1.0
Content-type: text/html
Last-modified: Wed, 14-Mar-95 18:15:23 GMT
Content-length: 95

<title>User Created</title>
<h1>The util-tester account has been created
</h1>
```

Figura 6: Respuesta HTTP desde el Servidor

Esto es un ejemplo de la respuesta que se puede obtener desde el servidor, al igual que en las peticiones se incluyen datos de control como la versión del HTTP usado, el tamaño del cuerpo que encapsula la respuesta o el tipo de contenido que se está mandando, pero además todas las respuestas HTTP incluyen un código de respuesta, que informa al usuario de cuál ha sido el resultado a priori de la solicitud hecha.

CÓDIGO DE RESPUESTA	SIGNIFICADO
100-199	Información
200-299	Petición procesada correctamente
300-399	Redirección
400-499	Petición incompleta/mal formulada
500-599	Error del servidor

Tabla 1: Códigos de Respuesta HTTP

2.2.2.2. Otros Protocolos

Aunque HTTP haya sido el protocolo usado en el entorno de pruebas desarrollado para este trabajo, en la actualidad se usan multitud de otros protocolos:

- **FTP** (File Transfer Protocol):

Protocolo de red pensado específicamente para la transferencia de archivos desde un servidor a un cliente con la máxima velocidad posible, sin embargo, FTP no es de los protocolos más seguros ya que transmite los archivos en texto plano.

- **SMTP** (Simple Mail Transfer Protocol):

Protocolo que es usado para gestionar el envío y recepción de correos electrónicos, también al igual que el resto de los protocolos mostrados, usando el modelo cliente-servidor usando TCP/IP en las conexiones y usualmente combinado con otros protocolos de correo como IMAP o POP.

- **SSH** (Secure Shell):

Protocolo centrado en el acceso a un servidor por medio de una conexión segura donde todos los datos que sean intercambiados estén previamente cifrados, mediante este protocolo es posible hacer sesiones por ejemplo del protocolo FTP solventando así sus problemas de seguridad

- **TFTP** (Trivial File Transfer Protocol):

Versión simplificada del protocolo FTP, comparte su uso pues es utilizado para obtener o subir archivos a un host remoto, sin embargo, su implementación es más sencilla de llevar a cabo

3. Diseño

En esta sección se exploran las distintas posibilidades que se consideraron antes de comenzar a desarrollar el proyecto, así como las valoraciones que llevaron al resultado final, además se proporciona un entendimiento general de cómo se han organizado los distintos módulos creados y cómo funciona cada uno de ellos visto desde un punto de vista teórico.

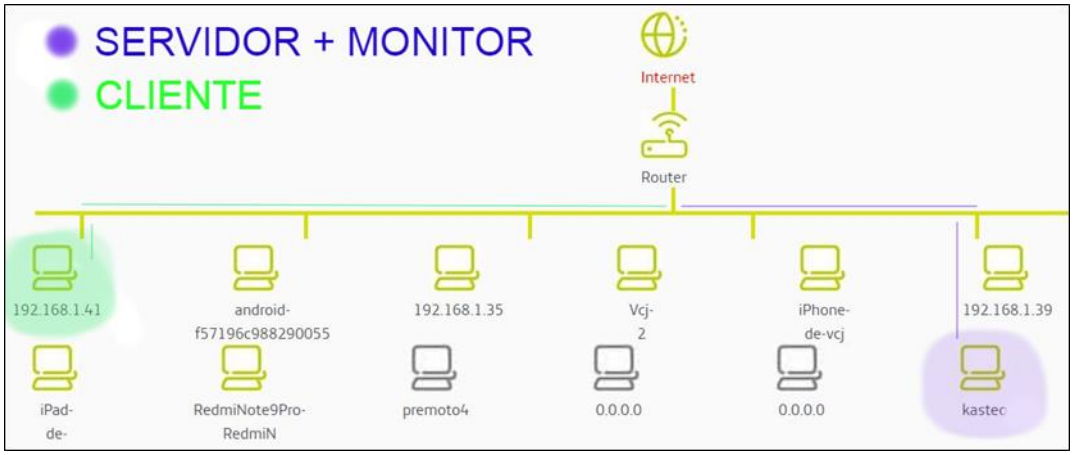

Figura 7: Disposición de la Red Local

Para una primera vista superficial se plantea la estructura mostrada en la figura anterior, en ella se aprecia cómo se han dispuesto las distintas partes que componen el total del código desarrollado a lo largo del trabajo. Se pueden distinguir tres principales partes, todas ellas serán descritas en profundidad a lo largo de esta sección, así como de qué manera se han coordinado para conseguir los objetivos buscados.

3.1. Creación del Servidor

En un primer momento fue planteada la idea de desarrollar un servidor que actuase como agente intermedio entre un cliente y una base de datos, para así controlar las peticiones y las consultas que fueran realizadas.

La idea base del proyecto era desarrollar el servidor usando el lenguaje de programación Java, idea razonable teniendo en cuenta que es uno de los lenguajes más usados para estas funciones junto con PHP y Python entre otros. El motivo es que es un lenguaje escalable y fácil de mantener, además de que cuenta con numerosas librerías siendo un lenguaje open-source que hacen que sea relativamente sencillo el implementar funciones en los servidores. Sin embargo, cabe mencionar que aun así se planteó el usar otros lenguajes, pero finalmente no fue llevado a cabo.

Tras asegurar que el lenguaje a utilizar iba a ser Java, el siguiente paso fue elegir la estructura que iba a seguir el servidor, pues existen distintos tipos de servidores según su funcionamiento interno y su manera de contestar las peticiones de los clientes.

3.1.1. Servidor de hilo único

Es un tipo de servidor muy básico y realmente poco práctico en el que las peticiones de los clientes son atendidas por el mismo hilo que acepta y establece la conexión. Esto es una mala idea debido a que el servidor solo será capaz de atender nuevas peticiones siempre y cuando esté en la función serverSocket.accept() la cual se encarga de esperar hasta que haya un intento de establecer una conexión y eso teniendo en cuenta que a mayor sea el tiempo que tienen que esperar los clientes para ver sus conexiones aceptadas, mayor será la probabilidad de que sus peticiones sean denegadas es una idea muy poco eficiente.

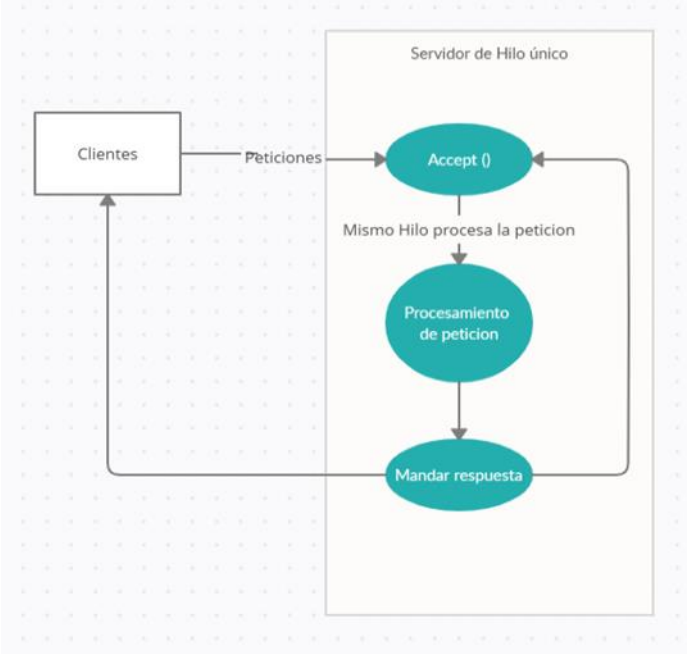

Figura 8: Esquema Servidor, modelo hilo único

3.1.2. Servidores multi-hilo

Este tipo de hilo aun con sus desventajas solventa los problemas presentados por los servidores de hilo único, la mayor diferencia respecto al anterior es que cuando una petición nueva llega al servidor, se lanza un hilo nuevo que atiende al cliente, por tanto, se logran dos objetivos:

- En caso de que la petición recibida requiera de bastante tiempo para ser atendida correctamente, el servidor no estará fuera de la función serverSocket.accept() con lo que podrá seguir atendiendo nuevas peticiones.
- La única forma de bloquear el servidor sería recibir una petición que consumiera la mayor parte de la CPU y/o el ancho de banda de conexión del servidor, algo altamente improbable

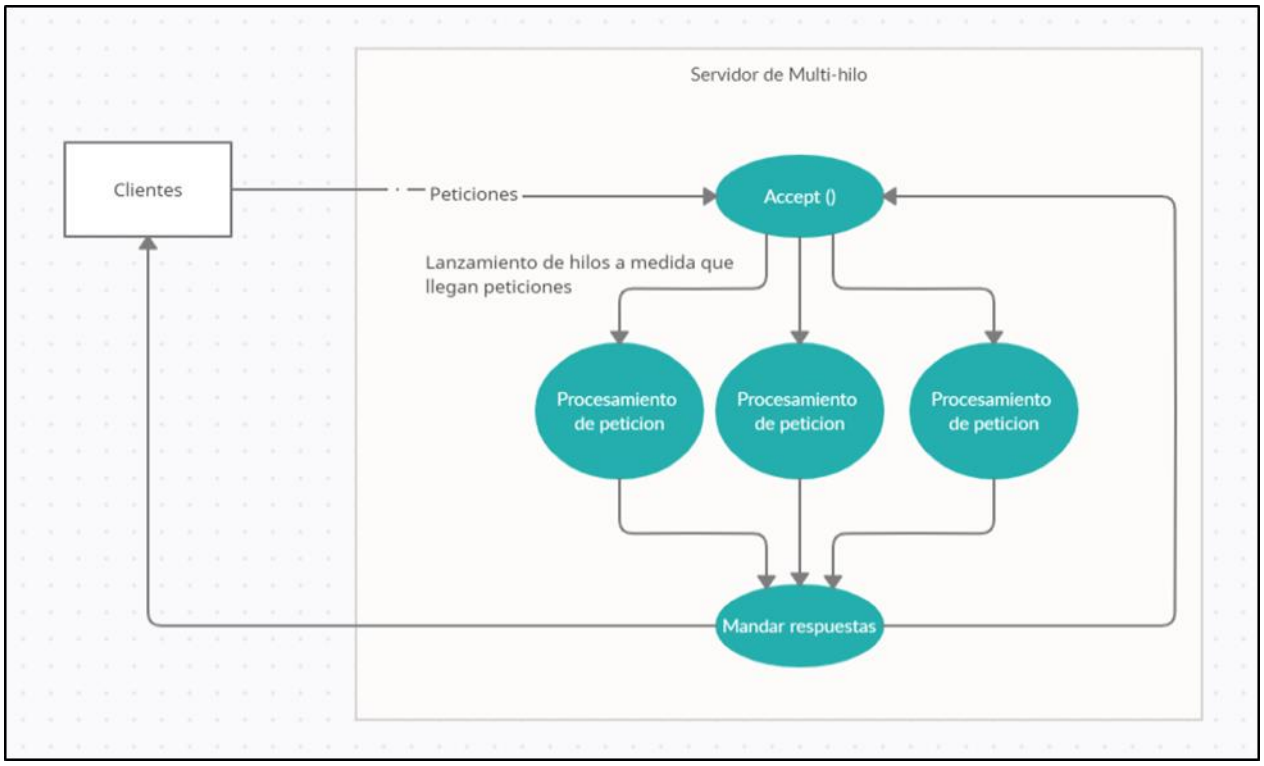

Figura 9: Esquema Servidor, modelo multi-hilo

3.1.3. Servidores con pool de hilos

Finalmente, este tipo de servidor es el que ha sido elegido para el desarrollo del trabajo, la idea básica es continuar con las ventajas que proporciona el servidor multi-hilo, pero además proporcionando un mayor control de los recursos del servidor a la hora de gestionarlo.

Al igual que el servidor multi-hilo, estos servidores disponen de hilos que procesan las peticiones de forma paralela mientras el servidor sigue escuchando por nuevas solicitudes, pero con la diferencia de que en un supuesto caso de tráfico elevado, el servidor multi-hilo lanza tantos hilos paralelos como sean requeridos para atender las peticiones hecho que podría llegar a colapsar un servidor si no se trata con cuidado, en cambio un servidor con un pool de hilos gestiona este problema de manera más estática, pues el número máximo de conexiones paralelas son establecidas desde el comienzo de la ejecución del servidor.

Esto puede representar un problema pues en caso de que el número de solicitudes recibidas sea mayor que el número de hilos, multitud de solicitudes se verán desatendidas, pero como consecuencia el servidor podrá trabajar al límite de su capacidad de procesamiento sin llegar a verse desbordado y colapsar.

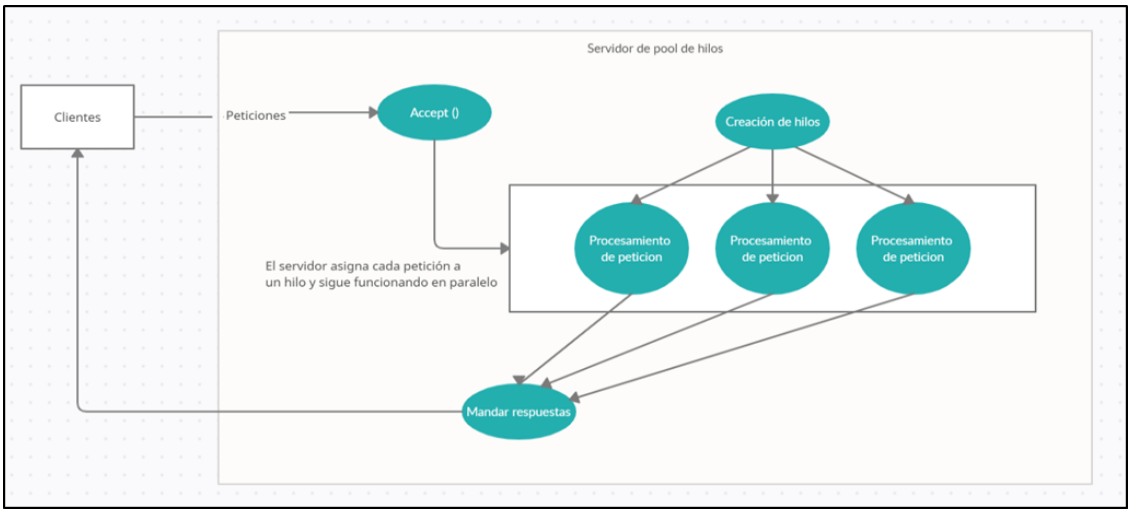

Figura 10: Esquema Servidor, modelo pool de hilos

3.2. Creación del Cliente

La primera idea era la de crear un usuario correctamente, que estableciera la conexión TCP/IP al servidor y realizase una acción básica y existen múltiples formas de hacerlo:

- Interacción con el servidor de forma manual

En caso de que se hubiera implementado en el servidor una funcionalidad Front-end, donde los clientes pudieran interactuar y por ejemplo pulsando un botón mandar una petición HTTP/GET se podría probar fácilmente y de forma visual si las conexiones se establecen correctamente, sin embargo, este método es el más ineficiente a la hora de probar la eficiencia del servidor en cuanto a la gestión de una gran cantidad de solicitudes.

- Establecimiento de peticiones automatizadas

Existen múltiples programas ya creados que permiten automatizar el envío de peticiones a un servidor como puede ser JMeter sin embargo su uso fue descartado para entrar más en detalle en la cantidad de solicitudes que se envían, con qué frecuencia, la composición de estas, el establecimiento de las conexiones, etc.

Otra opción dentro de la automatización es crear un programa que simule el envío de peticiones y múltiples opciones se plantean, librerías de sockets en cualquier lenguaje, módulos ya implementados como requests de Python, comandos de terminal como cURL, nC, telnet.

El resultado final ha sido el resultado de juntar dos tecnologías, como base para establecer las conexiones se ha utilizado el comando cURL, este es un comando propio de Unix que automatiza el envío de datos además de proveer el uso de varios protocolos entre ellos HTTP que es el soportado por el servidor desarrollado.

Para la paralelización de las peticiones de manera controlada ha sido desarrollado un script en el lenguaje Python mediante su librería Threading. La primera versión

sencillamente creaba 10 hilos que de forma continua enviaban peticiones hacia la dirección IP:Puerto en el que estuviera establecido el servidor.

Esta versión fue suficiente para probar que el servidor era capaz de procesar solicitudes y contestar de forma correcta, además la información proporcionada por cURL ha sido de gran ayuda para comprobar errores en cuanto a la formulación de la respuesta desde el servidor siguiendo el protocolo HTTP de forma correcta.

Sin embargo, se plantea una problemática a la hora de usar este programa como una representación real y es que como es razonable ningún servidor experimenta una cantidad de tráfico de forma constante e igual por lo tanto se plantearon varias alternativas sin las cuales el resultado de analizar el tráfico enviado al servidor hubiera sido parecido a lo expuesto a continuación.

Figura 11: Representación tráfico irreal de un servidor

Por lo tanto, la primera solución fue la de generar un número aleatorio y variable de usuarios que realizaran una cantidad fija o también aleatoria de peticiones, esta solución podría haber sido suficiente sin embargo tampoco parece realista que en un momento dado haya por ejemplo 500 peticiones y un segundo después ninguna. La solución óptima encontrada fue la de generar un ruido, que se comportase de manera pseudoaleatoria, es decir sus valores no son completamente deterministas pero la relación entre ellos sigue una tendencia [8].

$$W(t) = \int_0^t \frac{dW(\tau)}{d\tau} d\tau$$

Figura 12: Fórmula del Ruido Browniano o proceso Wiener donde se define como la integral a lo largo del tiempo de un ruido blanco

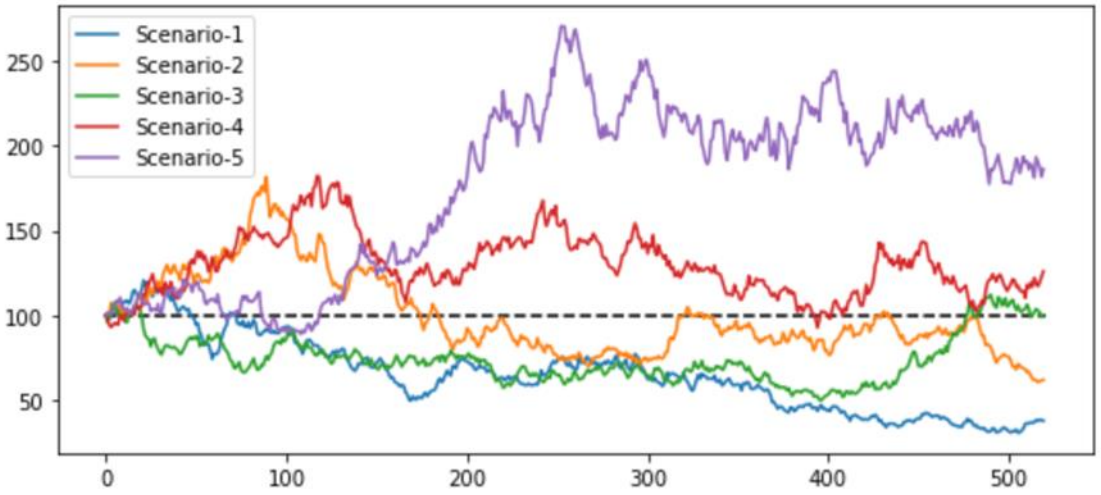

Figura 13: Representación visual de ruidos Brownianos

Gráfica extraída de [8] donde se muestra una simulación del precio de acciones en el mercado a lo largo de 52 semanas, con distintas volatilidades.

Por último, los valores de este ruido Browniano podían generar picos inesperadamente bruscos y valores negativos que no tendrían sentido, motivo por el cual se decidió como último paso aplicar una ecuación diferencial fraccionaria *"Fractional calculus is a generalization of the ordinary differentiation and integration to ar-bitrary non-integer order, it is a field of mathematic study that grows out of the traditionaldefinitions of the calculus integral and derivative operators in much the same way fractionalexponents is an outgrowth of exponents with integer value."* [22]

Este método es usado en otros campos para el análisis gráfico, pues aplicado de la forma correcta es capaz de transformar una gráfica en otra que mantenga tanto las tendencias como los picos que la definen, pero rebajándola para que sea más sencilla de interpretar, más adelante se explicara como han sido aplicados estos métodos pero como conclusión a este apartado se presentan varias representaciones que muestran el tráfico o la tendencia con la que el programa genera las peticiones que se realizan al servidor.

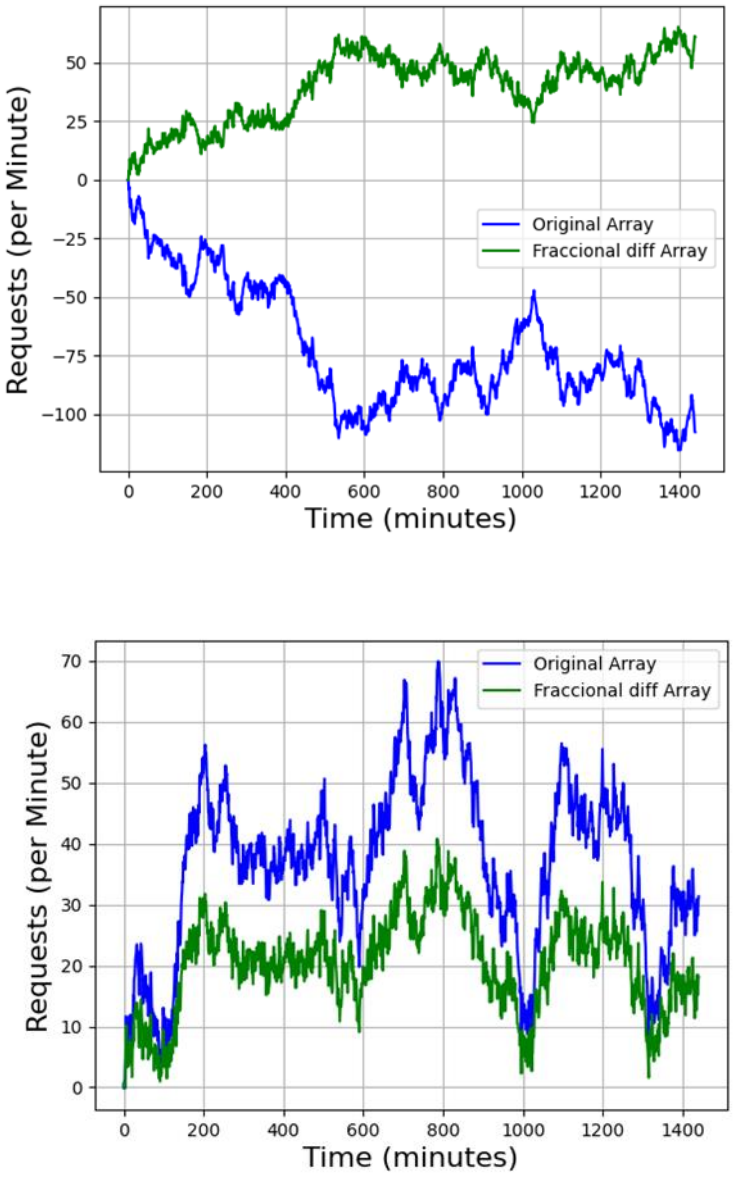

Figura 14: Gráficas representativas de la creación de usuarios en el cliente

Ambas gráficas han sido generadas automáticamente por el programa mediante el módulo matplotlib de Python, estas representan la tendencia de generación de usuarios, y en ambas se puede apreciar que las gráficas originales y las gráficas resultado de tratarlas son equivalentes.

Además, en la primera es apreciable que los valores negativos son procesados como valores absolutos para mantener el sentido.

3.3. Creación del Monitor

Como se ha explicado anteriormente, la idea inicial era monitorizar un servidor que hiciera las veces de conexión entre los usuarios y una base de datos, para lo cual se iba a utilizar un agente APM de Java denominado Elastic.

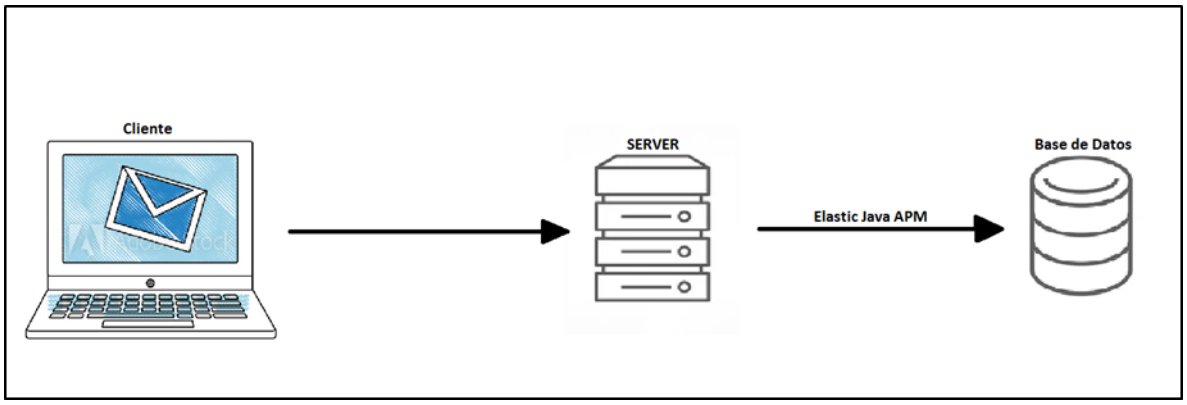

Figura 15: Esquema inicial agente APM Elastic search

Una vez los datos fueran recogidos serían representados con una de dos opciones, la herramienta de software libre Grafana desarrollada por Grafana labs o la herramienta Kibana desarrollada por Elastic NV.

Descripciones oficiales de cada herramienta dispuestas en sus páginas web oficiales:

- Grafana:

"Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored. Create, explore, and share dashboards with your team and foster a data driven culture."

- Kibana:

"Kibana is a free and open user interface that lets you visualize your Elasticsearch data and navigate the Elastic Stack. Do anything from tracking query load to understanding the way requests flow through your apps."

Ambas herramientas son similares pues su función es la de representar series de datos, mantener alertas de monitorización y ambas están sobre todo enfocadas a controlar servidores, sin embargo, hay diferencias que han determinado que en última instancia el software elegido haya sido Grafana.

1. En cuanto a la captura de datos, Kibana se ve limitado a la representación de datos obtenidos mediante los softwares propios de la empresa Elastic, que, aun ofreciendo una amplia flexibilidad, sigue siendo menor que Grafana siendo que esta última está preparada para usar múltiples y variadas tecnologías como MySQL o InfluxDB.

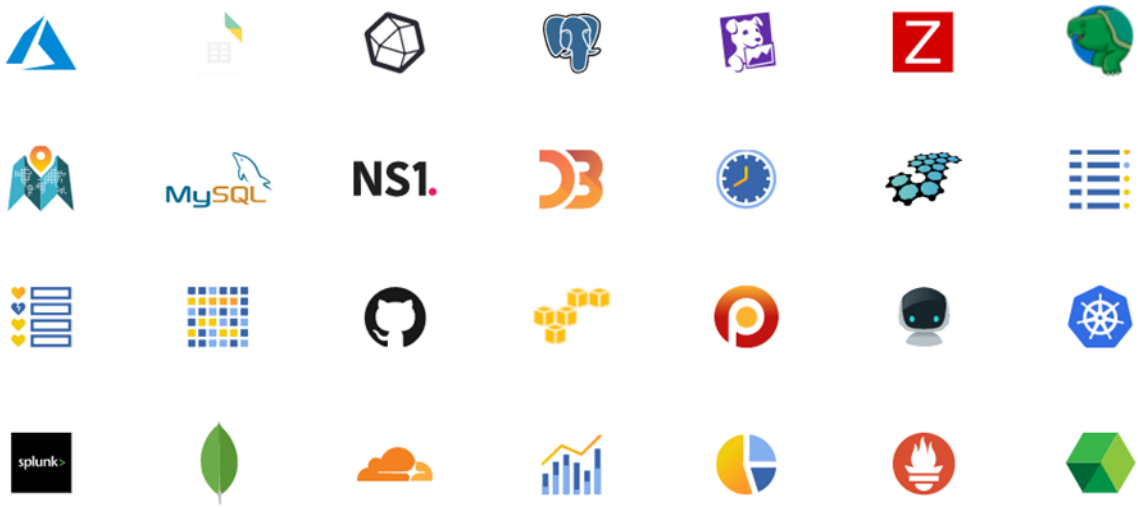

Figura 16: Tecnologías soportadas por Grafana

Tecnologías sincronizables con Grafana a través de Plugins

2. Grafana muestra una superioridad en cuanto al análisis de servidores per se, pues está más enfocado al análisis de estadísticas métricas (cpu, memoria, I/O) algo que ha sido determinante a la hora de desarrollar este trabajo, sin embargo, no cuenta con la capacidad de Kibana de analizar datos de texto más extensos o complejos como pueden encontrarse en labores de Big Data.

Como conclusión, la versatilidad de Grafana y su función principal de análisis de métricas, han hecho que sea la elección final para la representación de los datos obtenidos, pues los puntos fuertes de Kibana expuestos, así como que al ser un software desarrollado en JavaScript sea ejecutable en cualquier plataforma, han sido considerados como innecesarios o de poca importancia para la extracción de conclusiones buscadas como finalidad de este entorno de pruebas.

Tras determinar cómo van a ser representados los datos, solo queda exponer cómo estos han sido extraídos y guardados. Existen multitud de opciones, pero no ha sido difícil determinar que, por simplicidad, eficiencia y versatilidad, una buena combinación de recolector y almacenador de datos era Telegraf e InfluxDB, ambas herramientas pertenecientes a la empresa Influxdata.

- Telegraf

Telegraf es una herramienta de código libre que con más de 200 plugins, convierte la recolección de datos en una tarea realmente sencilla, en este proyecto se ha usado para recolectar las estadísticas básicas mencionadas anteriormente, capacidad de la CPU, uso de la memoria, uso del disco, etc.

- InfluxDB

InfluxDB es una base de datos realmente sencilla y eficiente, que se puede mantener en segundo plano activa como un proceso, haciendo las veces de contenedor para los datos que telegraf vaya recolectando, además de que la compatibilidad entre InfluxDB con Telegraf y Grafana es idónea y ha funcionado sin representar ningún problema a lo largo del desarrollo de las pruebas realizadas.

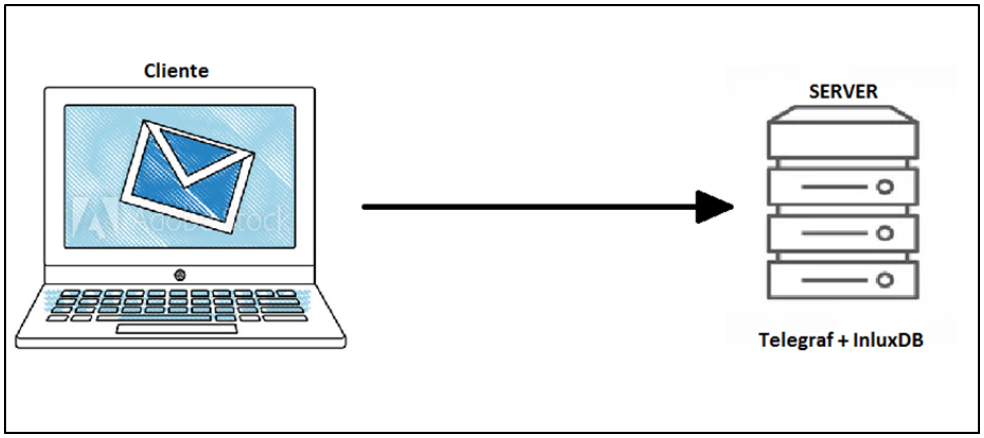

Figura 17: Esquema final de recolección de datos

3.4. Establecimiento de la Red/Conexiones

A la hora de ejecutar el software creado se ha intentado alcanzar una situación lo más parecida a la realidad dentro de ciertas limitaciones.

El primer problema planteado es como dejar al servidor ejecutarse en una máquina sin perjudicar los datos extraídos, pues en caso de hacer funcionar al servidor en el mismo hardware que crea y manda las peticiones, aparecen dos problemas, el primero es que al estar haciendo las peticiones a un host local el tiempo de retardo sería literalmente de 0 milisegundos, valor que es en esencia irreal cuando se hacen peticiones a través de internet, el segundo es que el proceso de crear y mandar peticiones también genera un costo en uso de los recursos del ordenador, por tanto los datos extraídos no serían especialmente representativos.

La solución a estos dos problemas planteó dos maneras para ser resuelta, una costosa como es el establecer el servidor en una máquina cuya dirección sea pública, es decir subir el servidor web a internet, pero esta opción es costosa y fue considerada un exceso de medios, por tanto y como alternativa para el desarrollo de las pruebas, el servidor ha sido ejecutado en un ordenador que ha permanecido activo durante 24 horas sin ninguna intervención y el programa que hace las veces de cliente ha sido ejecutado en un ordenador aparte pasando por un router que conectaba ambas máquinas.

Por lo tanto, las pruebas y el envío de peticiones han sido llevado a cabo a través de una red de área local, LAN (Local Area Network) pasando por distintos problemas que se han presentado y se explicarán más adelante como el bloqueo de conexiones realizado por el Firewall.

4. Desarrollo

4.1. Implementación del Servidor

Como ha sido indicado anteriormente, el servidor ha sido implementado usando el lenguaje Java, desarrollado usando el IDE Eclipse y probado dentro de este. La base para la creación del código ha sido sacada de [5] pero ha sido modificada para cumplimentar más aspectos interesantes y/o necesarios para el proyecto.

Cabe remarcar que para la ejecución del servidor de forma eficiente no tiene sentido que este se ejecute dentro de un IDE teniendo en cuenta los recursos que estos programas consumen, por lo tanto, el objeto final del desarrollo ha sido exportado en un archivo JAR para ser ejecutado de forma sencilla desde una terminal de comandos usando "java -jar server.jar".

Figura 18: Archivo ejecutable generado

El código fuente del servidor se compone de dos clases:

Clase PoolThreadedServer:

Esta clase se encarga de implementar el bucle principal del servidor que atiende las peticiones y se puede dividir su funcionalidad en dos partes principales.

```
public static void main(String[] args) {

    PoolThreadedServer server = new PoolThreadedServer(8080);

    new Thread(server).start();
    System.out.println("Server started...");
    try {
        Thread.sleep(Long.MAX_VALUE);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Stopping Server");
    server.stop();

}
```

Figura 19: Función principal del servidor

La función principal del servidor sencillamente crea una instancia de la clase y la ejecuta.

```
public void run(){
    synchronized(this){
        this.runningThread = Thread.currentThread();
    }

    getIPaddr();
    openServerSocket();

    Socket clientSocket = null;

    while(!isStopped()){

        clientSocket = null;

        try {
            clientSocket = this.serverSocket.accept();
        } catch (IOException e) {
            if(isStopped()) {
                System.out.println("Server Stopped.") ;
                break;
            }
            throw new RuntimeException(
                "Error accepting client connection", e);
        }
        requests++;
        System.out.println("Requests processed: " + requests);
        this.threadPool.execute(new WorkerRunnable(clientSocket, "Thread Pooled Server"));

    }

    try {
        clientSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }

    this.threadPool.shutdown();
    System.out.println("Thread Stopped.") ;
}
```

Figura 20: Bucle principal del servidor

Por otro lado, el bucle principal de la clase, implementado en la función de la clase run(), establece que el hilo principal que acepta las peticiones es el hilo creado en la función principal mostrada anteriormente, a partir de ahí y tras haber establecido la dirección IP y el puerto donde se va a disponer el servidor, de manera indefinida el hilo principal se dedica a aceptar peticiones, en caso de que haya algún problema estableciendo la conexión, o de que el servidor se quiera cerrar sencillamente se para el bucle principal. En caso contrario el bucle crea un nuevo hilo y este crea una instancia de la otra clase principal del servidor que se encarga de procesar la solicitud del cliente.

Clase WorkerRunnable:

Esta clase como ya se ha indicado se encarga de procesar las peticiones y también puede ser explicada en dos partes.

```
public void run() {
    try {
        String response = "Respuesta desde el servidor.\n\n";
        byte[] responseBytes = response.getBytes();
        InputStream input  = clientSocket.getInputStream();
        OutputStream output = clientSocket.getOutputStream();
        java.util.Date date = new java.util.Date();
        long timeBef = System.currentTimeMillis();
        long timeEnd;
        this.perderTiempo();
        this.IOuse();
        output.write(("HTTP/1.1 200 OK\nContent-Length: " + responseBytes.length + "\n\nRespuesta desde el servidor.\n\n").getBytes());
        output.close();
        input.close();
        timeEnd = System.currentTimeMillis();
        System.out.println(date.toString() + " | Request processed: " + (timeEnd - timeBef)/1000 +
                " seconds " + (timeEnd - timeBef)%1000 + " milliseconds.");

    } catch (IOException e) {

        e.printStackTrace();

    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Figura 21: Bucle principal de procesado de peticiones

La función principal de esta clase construye la respuesta que se va a mandar al cliente que ha formulado la petición, como se puede observar esta respuesta sigue el formato HTTP expuesto en apartados anteriores, pero de una manera sencilla albergando únicamente el protocolo, el código de respuesta, el tamaño del mensaje que se va a mandar de vuelta al cliente y el mensaje como tal.

De forma adicional esta clase realiza dos funciones más, la primera es la de dar información a la máquina que está ejecutando el servidor, capturando información como la fecha de procesado de la petición y el tiempo que ha tardado en ser procesada y la segunda es forzar un aumento del uso de la CPU calculando un numero factorial, una pérdida de tiempo voluntaria para simular un uso del servidor prolongado y generar un uso de las entradas y salidas, escribiendo y leyendo de un fichero de texto repetidas veces, todo apreciable en la siguiente figura.

```java
public void perderTiempo() throws InterruptedException {
    int i, res;
    for(i = 40, res = 1; i >= 1; i--) {
        res *= i;
    }
    TimeUnit.SECONDS.sleep(5);
}

public void IOuse() {

    int i;

    try {
        File IO = new File("IOfile.txt");
        if (IO.createNewFile()) {
          System.out.println("File created: " + IO.getName());
        }
    } catch (IOException e) {
            System.out.println("Error during file creation.");
            e.printStackTrace();
    }

    try {
        FileWriter myWriter = new FileWriter("IOfile.txt");
        for(i = 0; i < 100; i++) {
            myWriter.write("Lorem ipsum dolor sit amet, consectetur adipiscing elit,"
                    + " sed do eiusmod tempor incididunt ut labore et dolore magna aliqua."
                    + " Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris"
                    + " nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in"
                    + " reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur."
                    + " Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt"
                    + " mollit anim id est laborum.");
        }
        myWriter.close();
        new PrintWriter("IOfile.txt").close();
    } catch (IOException e) {
        System.out.println("Error during file edition.");
        e.printStackTrace();
    }
}
```

Figura 22: Funcionalidad añadida al servidor

4.2. Implementación del Cliente

Como ha sido explicado el cliente ha sido desarrollado con Python, en gran parte debido a la versatilidad que este lenguaje ofrece y además debido a que es suficientemente eficiente como para llevar a cabo el trabajo requerido. El módulo desarrollado se compone de dos partes principales.

Módulo BROWNIAN.py

Es el módulo que implementa la funcionalidad de las librerías importadas para la generación del ruido Browniano y la ecuación diferencial fraccionaria.

```python
def brownian(a, n, dt, delta, out=None):

    a = np.asarray(a)

    # Para cada elemento del array a, genera una muestra de n numeros en distribucion normal
    r = norm.rvs(size = a.shape + (n,), scale=delta*sqrt(dt))

    # Si no se proporciona un array de salida se genera uno
    if out is None:
        out = np.empty(r.shape)

    # Crea la tencendia browniana sumando la muestras aleatorias
    np.cumsum(r, axis=-1, out=out)

    # Anade la condicion incial.
    out += np.expand_dims(a, axis=-1)

    return out

def fracdif_brownian(x):
    return fdiff(np.absolute(x), n=0.2)
```

Figura 23: Funciones básicas de Brownian.py y Fracdiff.pay

Módulo requests.py

Este módulo es el programa principal de emulación de tráfico y se compone de tres partes principales:

- Creación de la tendencia de generación de clientes.

```
# The Wiener process parameter.
delta = 2

# Total time.
T = 24*60

# Number of steps.
N = 24*60

# Time step size
dt = T/N

# Create an empty array to store the realizations.
x = np.empty((1,N+1))
y = np.empty((1,N+1))

# Initial values of x.
x[:, 0] = 0
y[:, 0] = 0

BROWNIAN.brownian(x[:,0], N, dt, delta, out=x[:,1:])
y[0] = x[0]
t = np.linspace(0.0, N*dt, N+1)

y = BROWNIAN.fracdif_brownian(x)

plot(t, x[0], "-b", label="Original Array")
plot(t, y[0], "-g", label="Fraccional diff Array")
legend()
xlabel('Time (minutes)', fontsize=16)
ylabel('Requests (per Minute)', fontsize=16)
grid(True)
show()
```

Figura 24: Establecimiento de parámetros para el cliente

- Generación de clientes.

```
for i in range(y.size):


    print("Intervalo: ", i)
    print("Usuarios: ", int(y[0][i])*3)
    print("Peticiones por usuario: 5")
    print("Total de peticiones: ", int(y[0][i])*15)

    f.write("Intervalo: " + str(i))
    f.write("Usuarios: " + str(int(y[0][i])*3))
    f.write("Peticiones por usuario: 5")
    f.write("Total de peticiones: " + str(int(y[0][i])*15))
    f.write("\n\n")

    for j in range(int(y[0][i])*3):
        t = threading.Thread(target=curl)
        t.demon = True
        threadList.append(t)

    for j in range(int(y[0][i])*3):
        if not t.is_alive():
            threadList[j].start()

    for j in range(int(y[0][i])*3):
        threadList[j].join()

    threadList.clear()
```

Figura 25: Bucle principal del cliente

El método es bastante simple, el uso de la función print y write se ha puesto únicamente para controlar la creación de peticiones, en cuanto a la funcionalidad real, se aprecia como el programa recorre un bucle del tamaño del array generado, el cual tiene 1441 entradas (número de minutos en un día más uno). Para cada iteración del bucle, se generan n*3 usuarios, siendo n el numero extraído del array y cada usuario como se verá más adelante ejecuta 5 veces la petición al servidor, por eso para imprimir los datos de control el total de peticiones es n*15, finalmente los hilos creados se cierran y se borra su referencia de la lista de control de hilos.

- Ejecución del comando cURL hacia el servidor.

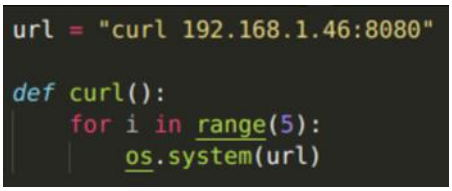

Figura 26: Establecimiento de conexión con el servidor por medio de cURL

4.3. Implementación del Monitor

Para la implementación y gestión de las herramientas de monitorización el proceso a resultado tan sencillo como descargar e instalar los tres programas elegidos, Telegraf, InfluxDB y Grafana.

Telegraf

Tras instalar el programa, es necesario antes de ejecutarlo llevar a cabo unas tareas de configuración previas, entre las cuales la más importante es la de crear el archivo telegraf.conf que sentará las bases de una correcta ejecución, estableciendo parámetros como que datos queremos que sean recogidos, donde queremos que sean guardados, etc.

En el caso de este proyecto, no se han usado plugins adicionales para captura de datos y estos han sido guardados en InfluxDB como se ha mencionado previamente.

InfluxDB

De nuevo el primer paso es descargar e instalar el programa tanto la parte de servidor denominada influxDB como la parte de cliente necesaria para probar el correcto funcionamiento denominada influx. Es necesario decir que esta tecnología usa un lenguaje propio para realizar las consultas denominado “Flux” pero este es relativamente sencillo y muy similar de base a SQL.

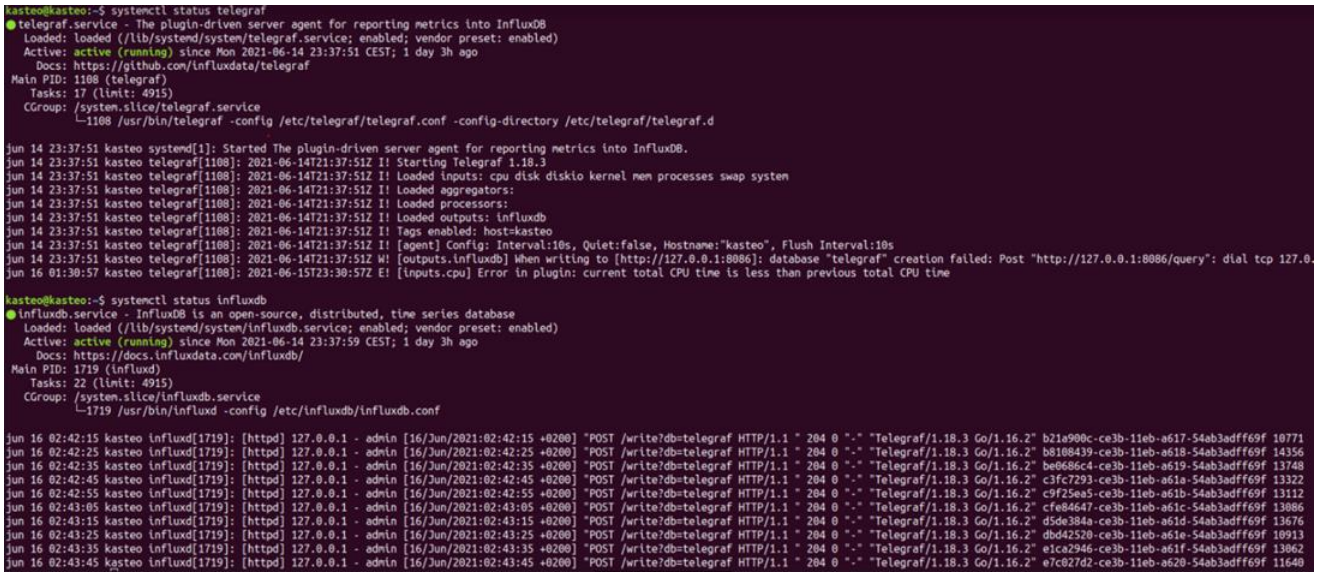

Figura 27: Estado activo de los procesos de Telegraf e influxDB

Ambos programas funcionan como servicios que funcionan de manera independiente, en la imagen puede apreciarse como tanto telegraf como influxDB están funcionando correctamente en la máquina donde se ejecutó el servidor de pruebas.

Teniendo esta situación presente ya se puede pasar a experimentar con la captura y almacenamiento de los datos, etapa previa a la representación de los mismos si la conexión entre nuestra base de datos y Grafana es satisfactoria.

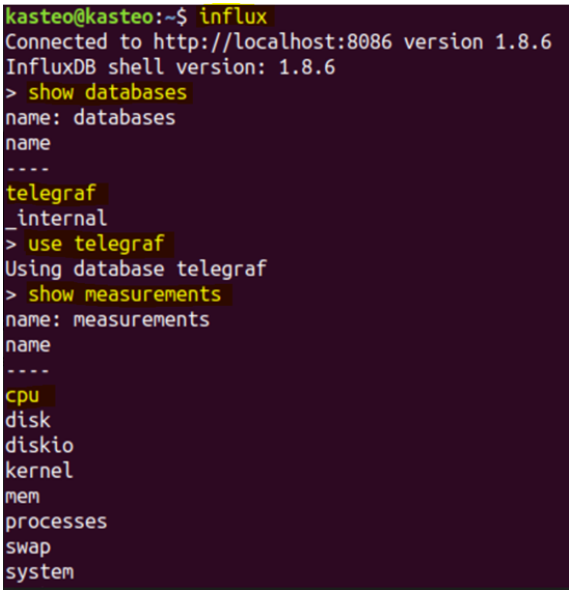

Figura 28: Uso básico de influxDB con Telegraf

En esta figura se puede apreciar cómo funciona el cliente de influxDB, además de que es un caso real que muestra cómo se han guardado los datos recopilados por telegraf, por simplicidad la base de datos ha sido llamada con el mismo nombre que el recolector y

mediante los comandos resaltados podemos ver que tablas se han creado en la base de datos, acceder a la información guardada es tan sencillo como realizar simples consultas ejemplificadas a continuación.

```
> select * from cpu limit 1
name: cpu
time                cpu       host   usage_guest usage_guest_nice usage_idle
----                ---       ----   ----------- ---------------- ----------
1623077010000000000 cpu-total kasteo 0           0                94.83320792576396
```

Figura 29: Ejemplo de query con el lenguaje Flux

Esta figura muestra como con una simple consulta similar al lenguaje SQL, se pueden obtener los datos que guardan telegraf e influxDB en cada tabla.

Por último, el uso de Grafana es intuitivo llegados a este punto donde todo esta configurado, solo queda conectar la base de datos y empezar a representarlos de la manera que se crea más conveniente.

Por lo tanto y como se ha comentado, en primer lugar, seleccionamos la fuente de datos que vamos a utilizar en la representación:

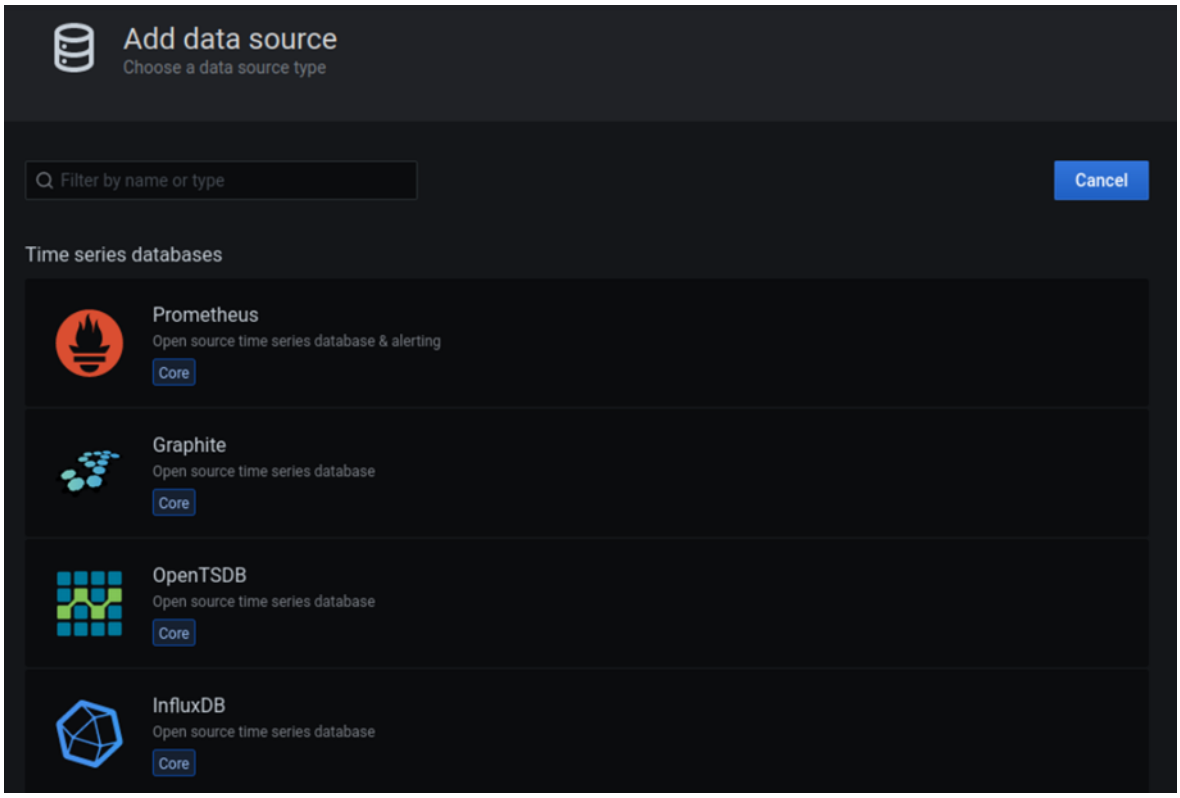

Figura 30: Selección de la Data Source para Grafana

En segundo lugar y si la configuración es llevada a cabo sin complicaciones, es el momento de comenzar a representar los datos, los cuales de nuevo son seleccionados

mediante consultas en el lenguaje propio de la base de datos elegida, aunque Grafana proporciona un método visual para elegir que datos queremos que sean representados.

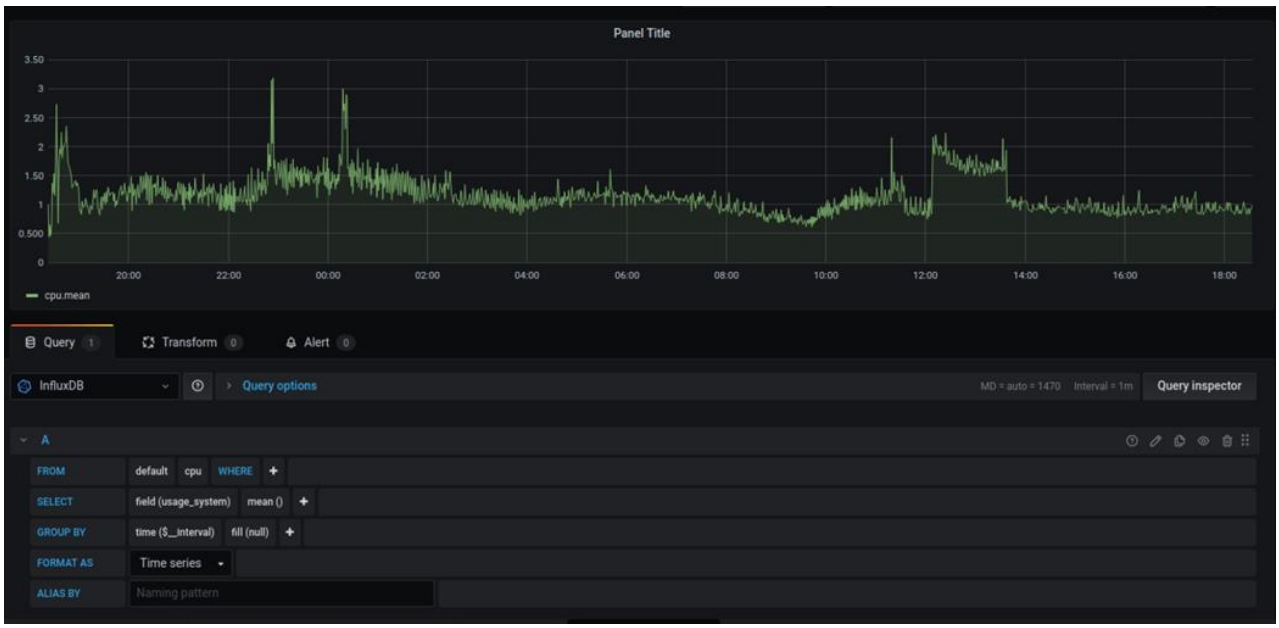

Figura 31: Ejemplo usando el método visual de consulta en Grafana

Figura 32: Ejemplo usando Flux para la consulta en Grafana.

5. Integración, pruebas y resultados

Llegando al final, aquí se exponen los resultados de combinar todo lo mencionado anteriormente y por tanto se mostrará el producto final creado, tanto los Dashboards generados con Grafana donde son apreciables los efectos del tráfico en un servidor, como los resultados directos obtenidos de ejecutar cada script creado para este entorno de pruebas.

El primer paso es, teniendo Telegraf e InfluxDB preparados, arrancar el servidor tras tenerlo desplegado en la red elegida y, habiendo abierto los puertos necesarios en el router que hace de intermediario, será necesario desactivar el Firewall para que la dirección sea accesible desde otros dispositivos dentro de la red local, una vez hecho esto, a continuación, se muestran los resultados de establecer conexiones con el servidor.

Caso 1: El servidor no ha sido desplegado correctamente

```
curl: (7) Failed to connect to 192.168.1.46 port 8080: Connection refused
```

Figura 33: Ejemplo de conexión fallida a través de cURL

Caso 2: El servidor ha sido desplegado y la petición es respondida satisfactoriamente

2.1 Uso del comando cURL con -v para ver la respuesta entera desde requests.py

```
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 30
<
Respuesta desde el servidor.

* Connection #0 to host 127.0.0.1 left intact
* Rebuilt URL to: 127.0.0.1:8080/
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 30
<
Respuesta desde el servidor.
```

Figura 34: Ejemplo de dos conexiones exitosas

2.2 Vista desde el servidor después de 24 horas procesando peticiones

```
Tue Jun 08 20:00:17 CEST 2021 | Request processed: 5 seconds 1 milliseconds.
Tue Jun 08 20:00:17 CEST 2021 | Request processed: 5 seconds 1 milliseconds.
Tue Jun 08 20:00:17 CEST 2021 | Request processed: 5 seconds 1 milliseconds.
Tue Jun 08 20:00:17 CEST 2021 | Request processed: 5 seconds 1 milliseconds.
Tue Jun 08 20:00:17 CEST 2021 | Request processed: 5 seconds 1 milliseconds.
Tue Jun 08 20:00:17 CEST 2021 | Request processed: 5 seconds 2 milliseconds.
Tue Jun 08 20:00:17 CEST 2021 | Request processed: 5 seconds 2 milliseconds.
Tue Jun 08 20:00:17 CEST 2021 | Request processed: 5 seconds 2 milliseconds.
Requests processed: 916657
Requests processed: 916658
Requests processed: 916659
Requests processed: 916660
Requests processed: 916661
Requests processed: 916662
Requests processed: 916663
Requests processed: 916664
```

Figura 35: Registro del servidor tras 24 horas de muestreo

Como se puede apreciar, el servidor maneja las peticiones correctamente, mostrando información detallada por cada una recibida, así como el cliente también muestra la respuesta recibida por parte del servidor, aunque sea una respuesta simple nos sirve para ver lo que queremos, la conexión se está estableciendo correctamente y las respuestas están llegando a través de la conexión sin problemas.

En última instancia y para obtener los resultados esperados, se expone el dashboard final generado con Grafana. En este dashboard podemos observar 4 gráficos que pueden ser interesantes a la hora de manejar datos de funcionamiento de un servidor.

El primero es un gráfico de control que muestra la cantidad de hilos existentes que ha habido a lo largo de la ejecución del servidor, la cual tuvo lugar entre el 6 y el 7 de Junio de 2021 durante alrededor de 24 horas, puede ser un valor relevante para que el dueño del servidor decida que se están generando demasiados hilos y evitar que esta situación genere lentitud en el servidor o malos funcionamientos, lo mismo con el segundo gráfico, que muestra el uso de la CPU, valor que es necesario controlar pues podría llegar a ser determinante para la estabilidad del servidor, por suerte Grafana es una herramienta que permite establecer límites y alertas para controlar estos factores que pueden representar riesgos tan grandes.

El segundo gráfico muestra el porcentaje de la memoria del servidor que está ocupado y en este se puede observar con color rojo un rango de peligro denominado threshold, si el valor registrado llegase a superar este threshold se podrían tomar acciones para mantener la integridad del servidor. El último gráfico sin ser tan relevante muestra la cantidad de lecturas y escrituras que se han realizado durante el lapso registrado.

Figura 36: Dashboard final obtenido con Grafana

Para una observación más precisa:

Figura 37: Uso de la CPU registrado durante el muestreo

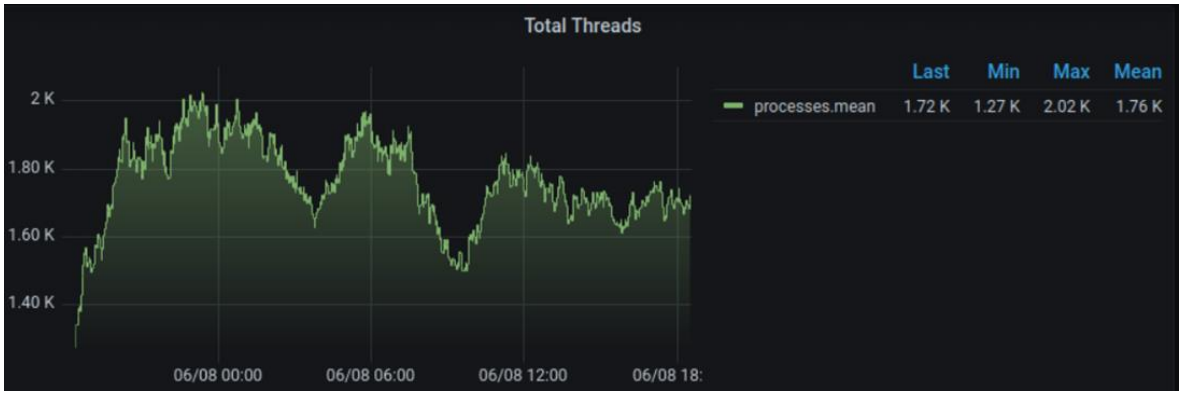

Figura 38: Cantidad de hilos creados durante el muestreo

6. Conclusiones y trabajo futuro

6.1. Conclusiones

Ha sido un trabajo realmente interesante, siendo un tema que como ya se ha expuesto a lo largo del documento, afecta de manera tan cercana a todos los usuarios de internet. Experimentamos caídas casi a diario de servicios, que nunca van a poder ser completamente evitadas, pero con una buena gestión e inversión en este tipo de procesos, pueden asegurar que sean lo más ocasionales posibles.

Se destaca en este documento la importancia de que no se ha inventado de nuevo la rueda si no todo lo contrario, solo se ha usado una combinación de las muchas que puede haber para controlar estos problemas y además se ha tratado de explicar por qué razones ha sido elegida cada tecnología usada y como resultado final, todo parece haber salido razonablemente bien, habiendo logrado los objetivos propuestos, creando un software capaz de guardar datos a tiempo real y visualizarlos de manera clara para ayudar en labores de seguridad en servidores.

En concreto en la simulación realizada se puede apreciar como a lo largo de las 24 horas, se experimentan diversos picos en valores como el uso de la CPU, al haber experimentado un tráfico aleatorio y siendo que el servidor no ha sido modificado ni afectado por ningún factor externo durante el intervalo de tiempo de la medición, es razonable pensar que los picos se deben a acumulaciones de "usuarios" creados por la simulación que de manera más o menos brusca han afectado al servidor, eso sí, sin llegar a tumbarlo, no porque este sea especialmente eficiente, sino porque ha sido programado para llevar a cabo funciones simples y rápidas.

6.2. Trabajo futuro

A nivel personal creo que es realmente motivador el haber usado tantas herramientas que están a la orden del día en cuanto a tecnologías se refiere, sin contar con que el proceso de aprender y experimentar en solitario con todo el tema referido a redes y conexiones ha sido realmente útil para afianzar contenidos impartidos durante la carrera y que son realmente útiles e importantes hoy en día en nuestra sociedad.

Razones por las cuales no descarto seguir estudiando las redes de aquí en adelante y si todo fuera bien dedicarme a ello tras acabar estudios más avanzados.

El software desarrollado parece fiable y reproducible y sienta unas bases donde multitud de modificaciones podrían llevarse a cabo que aumentarían las posibilidades ofrecidas, una buena opción por ejemplo sería la de añadir nuevos plugins a telegraf para la captura de más datos que puedan resultar interesantes, o en cuanto a la estructura del servidor, una mejora del propio servidor o una extensión de sus capacidades añadiéndole una base de datos sería algo interesante a valorar y digno de ser analizado.

7. Referencias

[1] – Mediacloud, *Grafana vs Kibana: ¿cuáles son las diferencias y cuál es mejor?* https://blog.mdcloud.es/grafana-vs-kibana-diferencias/

[2] – Elastic, *Monitoring Java applications with Elastic: Getting started with the Elastic APM Java Agent*: https://www.elastic.co/blog/monitoring-java-applications-and-getting-started-with-the-elastic-apm-java-agent

[3] Elastic - *APM Java Agent Set-up* : https://www.elastic.co/guide/en/apm/agent/java/current/setup.html

[4] Albert Coronado - *Monitorización de servidores con Grafana, Telegraf e InfluxDB*: https://www.youtube.com/watch?v=GLE71pIHUU8

[5] Jenkov.com, *Multithreaded servers in java*: http://tutorials.jenkov.com/java-multithreaded-servers/thread-pooled-server.html

[6] Castaño Díaz, Vicente José, *"Caracterización de servidores Web de ámbito académico,"* Universitat Politècnica de València, 2011.

[7] Baeldung – *A guide to Java sockets*: https://www.baeldung.com/a-guide-to-java-sockets

[8] Towards Data Science - *Brownian motion with Python*: https://towardsdatascience.com/brownian-motion-with-python-9083ebc46ff0

[9] PyPi.org - *Fracdiff: Super-fast Fractional Differentiation*: https://pypi.org/project/fracdiff/

[10] Scipy-cookbook – *Brownian Motion*: https://scipy-cookbook.readthedocs.io/items/BrownianMotion.html

[11] De Prado, M. L. (2018). *Advances in financial machine learning*. John Wiley & Sons.

[12] Navarro, Henry. (2015). *Representación del movimiento browniano fraccionario a partir de la ecuación del calor estocástica.* ***http://saber.ucv.ve/handle/123456789/12416***

[13 Janusz Gajda & Rafał Walasek, 2020. *"**Fractional differentiation and its use in machine learning**,"* Working Papers 2020-32, Faculty of Economic Sciences, University of Warsaw.

[15] Visual Capitalist. *Ranking the Top 100 Websites in the World*: https://www.visualcapitalist.com/ranking-the-top-100-websites-in-the-world/

[16] Blogthinkbig.com - Las mayores caídas de servicio de la historia: https://blogthinkbig.com/caidas-de-servicio

[17] Downdetector: https://downdetector.es/

[19] H. Yan, Y. Jiang and X. Zhou, *"A Bidirectional Chord System Based on Base-k Finger Table,"* 2008 International Symposium on Computer Science and Computational Technology, 2008, pp. 384-388, doi: 10.1109/ISCSCT.2008.55.

[20] C. Wong, HTTP pocket reference /, First edition. 2000.

[21] Seanwasere ytbe. *Grafana, Install Telegraf and Configure for InfluxDB:* https://www.youtube.com/watch?v=FrqeG-IajWM

[22] (American Psychological Assoc.)

Abbas, S., N'Guerekata, G. M., & Benchohra, M. (2014). *Advanced Fractional Differential and Integral Equations*. Nova Science Publishers, Inc.